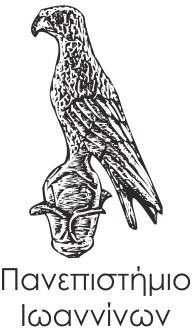
**ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ**



**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΑΛΓΟΡΙΘΜΟΙ ΚΑΙ ΠΡΟΧΩΡΗΜΕΝΕΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ**

**Διερεύνηση προσεγγίσεων επίλυσης του προβλήματος 0-1 σακιδίου**

ΤΕΧΝΙΚΗ ΕΚΘΕΣΗ

**ΛΟΥΠΑΣ ΒΑΣΙΛΕΙΟΣ**

**ΑΜ: 59**

Ημερομηνία Υποβολής: 7/1/2020

**1. ΤΟ ΠΡΟΒΛΗΜΑ**

Διερεύνηση προσεγγίσεων επίλυσης του προβλήματος 01σακιδίου (0/1 Knapsack)

Το πρόβλημα του 0/1 σακιδίου (01 knapsack) αφορά ένα σύνολο από αντικείμενα για τα οποία γνωρίζουμε το βάρος και την αξία κάθε αντικειμένου.

Ζητείται η επιλογή ενός υποσυνόλου των αντικειμένων έτσι ώστε το συνολικό βάρος από τα επιλεχθέντα αντικείμενα να μην ξεπερνά μια συγκεκριμένη τιμή βάρους και ταυτόχρονα να επιτυγχάνεται η μεγαλύτερη δυνατή αξία.

Το πρόθεμα 0/1 στο όνομα του προβλήματος υποδηλώνει ότι κάθε αντικείμενο μπορεί είτε να επιλεχθεί είτε να μην επιλεχθεί στο σύνολό του και όχι τμηματικά.

**2. ΠΡΟΣΕΓΓΙΣΕΙΣ ΕΠΙΛΥΣΗΣ - ΜΕΘΟΔΟΛΟΓΙΑ**

Για να διερευνήσουμε τις προσεγγίσεις επίλυσης του προβλήματος που μας έχει ανατεθεί πρέπει πρώτα να φτιαχτούν τα αρχεία (στιγμιότυπα) των προβλημάτων με τον generator που μας δόθηκε. Κατεβάζουμε το generator.c από τη διεύθυνση <http://hjemmesider.diku.dk/~pisinger/generator.c> και το κάνουμε compile (g++ -o generator generator.c) σε ένα terminal στο Linux.

Φτιάχνουμε ένα Linux bash file **(makefiles)** το οποίο εκτελούμε και παράγουμε τα 320 αρχεία με βάση την ονοματολογία που μας ζητήθηκε. Τα αρχεία αυτά έχουν την μορφή που παρουσιάζεται παρακάτω:

10

1 4 4

2 46 46

3 48 48

4 19 19

5 4 4

6 20 20

7 13 13

8 50 50

9 28 28

10 6 6

79

Το πρώτο νούμερο αναφέρεται στο πλήθος των αντικειμένων που μελετούνται, η πρώτη στήλη είναι ο Αύξων Αριθμός του αντικειμένου, η δεύτερη στήλη είναι η αξία του αντικειμένου και η τρίτη στήλη είναι το βάρος του. Το νούμερο της τελευταίας γραμμής φανερώνει τη συνολική χωρητικότητα του σακιδίου.

Έπειτα φτιάχτηκαν δυο προγράμματα σε c++ που ανοίγουν τα αρχεία αυτά ένα-ένα και καλούν functions οι οποίες υλοποιούν διαφορετικές μεθόδους/αλγόριθμους επίλυσης. Συγκεκριμένα τους:

* **Dynamic Programming**

Οι Αλγόριθμοι Δυναμικού προγραμματισμού βασίζονται στην επίλυση υποπροβλημάτων για να επιλύσουν το αρχικό πρόβλημα. Εφαρμόζουν στην τεχνική της βελτιστοποίησης και κάθε φορά που βρίσκεται λύση σε κάποιο υποπρόβλημα, αποθηκεύεται σε έναν πίνακα στον οποίο ο αλγόριθμος θα καταφεύγει κάθε φορά που θα συναντά το συγκεκριμένο πρόβλημα.

* **Greedy Approach**

Οι άπληστοι (Greedy) αλγόριθμοι, κάνουν πάντα την επιλογή που φαίνεται καλύτερη κάθε δεδομένη στιγμή.

* **Brute Force**

Οι αλγόριθμοι Brute Force δεδομένου ενός προβλήματος, εξετάζουν μία-μία όλες τις πιθανές λύσεις του προβλήματος για να βρουν τη βέλτιστη λύση**.** Κατά κανόνα τα συνδυαστικά προβλήματα έχουν ένα πολύ μεγάλο (εκθετικό) αριθμό πιθανών λύσεων και επομένως η τεχνική αυτή δεν μπορεί να λύσει στιγμιότυπα πέρα από κάποιο μέγεθος.

* **Branch and Bound**

Οι αλγόριθμοι διακλάδωσης και οριοθέτησης (Branch and Bound) έχουν σχεδιαστεί για προβλήματα βελτιστοποίησης και η βασική τους ιδέα είναι η απαλοιφή τμημάτων του χώρου αναζήτησης όπου γνωρίζουμε πως δεν υπάρχει λύση του προβλήματος.

* **OR-TOOLS**

Χρησιμοποιείται ο ενσωματωμένος επιλυτής που παρέχει η Google για το πρόβλημα του 0/1 σακιδίου. Knapsack**.**

Οι αλγόριθμοι αυτοί επιστρέφουν σαν αποτέλεσμα την αξία (value) την οποία υπολόγισε η κάθε μέθοδος, καθώς και τον χρόνο που χρειάστηκε (σε nanoseconds οι 4 πρώτες και millisecond η OR-TOOLS) ώστε να παράγουν το αποτέλεσμα.

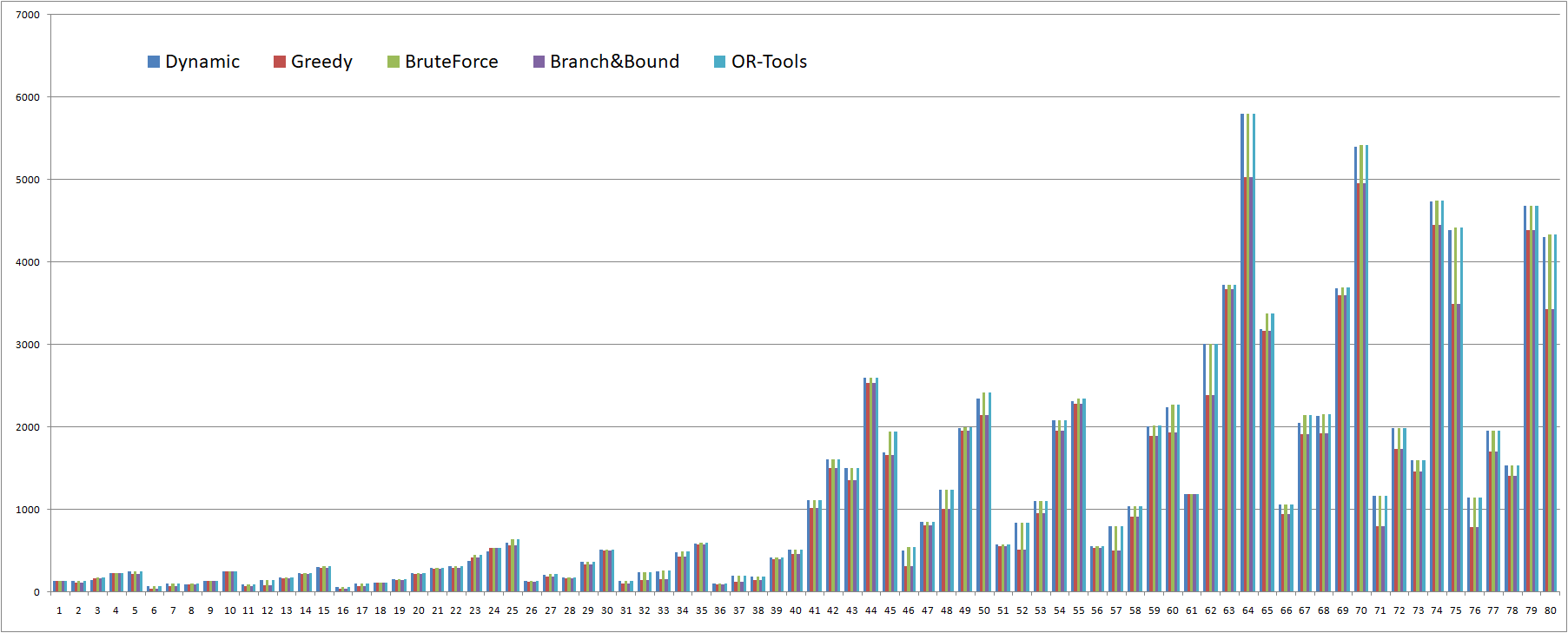
Οι κώδικες **Knapsack-Code.cc** και **ORTOOLS-Knap.cc** όταν εκτελεστούν παράγουν από ένα .csv αρχείο με το όνομα **results.csv** και **or-results.csv.**

Έπειτα τα αρχεία .csv ενώθηκαν και εισάχθηκε σε ένα book του Excel και προέκυψαν ραβδογράμματα για τις αξίες των λύσεων που μας έδωσαν οι 5 διαφορετικές προσεγγίσεις. Τα ραβδογράμματα αυτά προέκυψαν ομαδοποιώντας τα αποτελέσματα ανά αρχείο στιγμιότυπων (Πρώτα τα στιγμιότυπα με ν=10, έπειτα με ν=50, έπειτα με ν=100 και τέλος με ν=500). Το κάθε ραβδόγραμμα περιέχει 80 στήλες οι οποίες χωρίζονται σε 5 τιμές (1 για κάθε μέθοδο).

**3. ΑΠΟΤΕΛΕΣΜΑΤΑ**

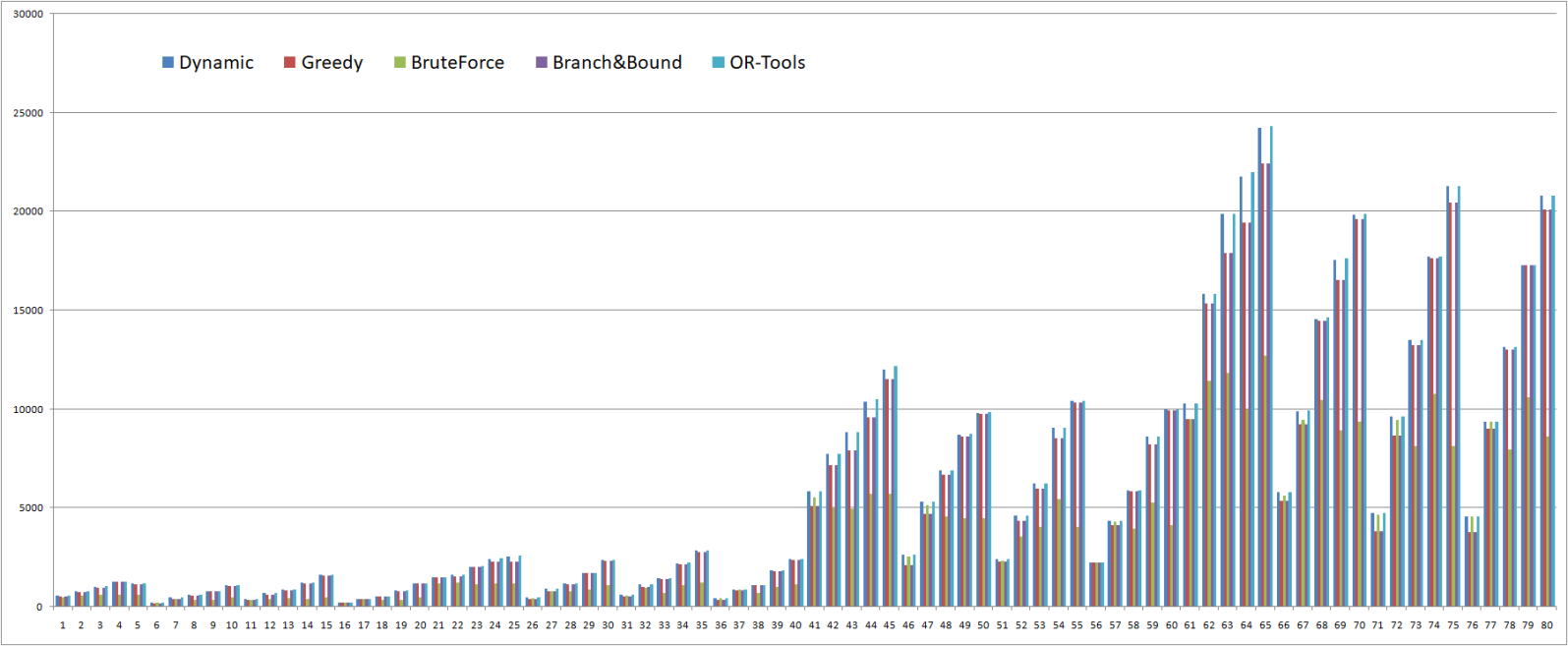
Παρατηρώντας τα αποτελέσματα εκτέλεσης των προγραμμάτων βλέπουμε ότι για μικρό αριθμό πλήθους αντικειμένων (ν=10) οι αλγόριθμοι αποδίδουν ταυτόσημα αποτελέσματα. Καθώς όμως αυξάνεται η συνολική χωρητικότητα σακιδίου οι αλγόριθμοι αρχίζουν να έχουν σοβαρές αποκλίσεις στο αποτέλεσμα με καλύτερα αποτελέσματα να δίνει ο επιλυτής OR-TOOLS και να ακολουθείται από τους Dynamic, και τον Brute Force (Ο οποίος δίνει πολύ ικανοποιητικά αποτελέσματα σε μικρό πλήθος συνδυασμών).

(Βλ. Γράφημα 1)



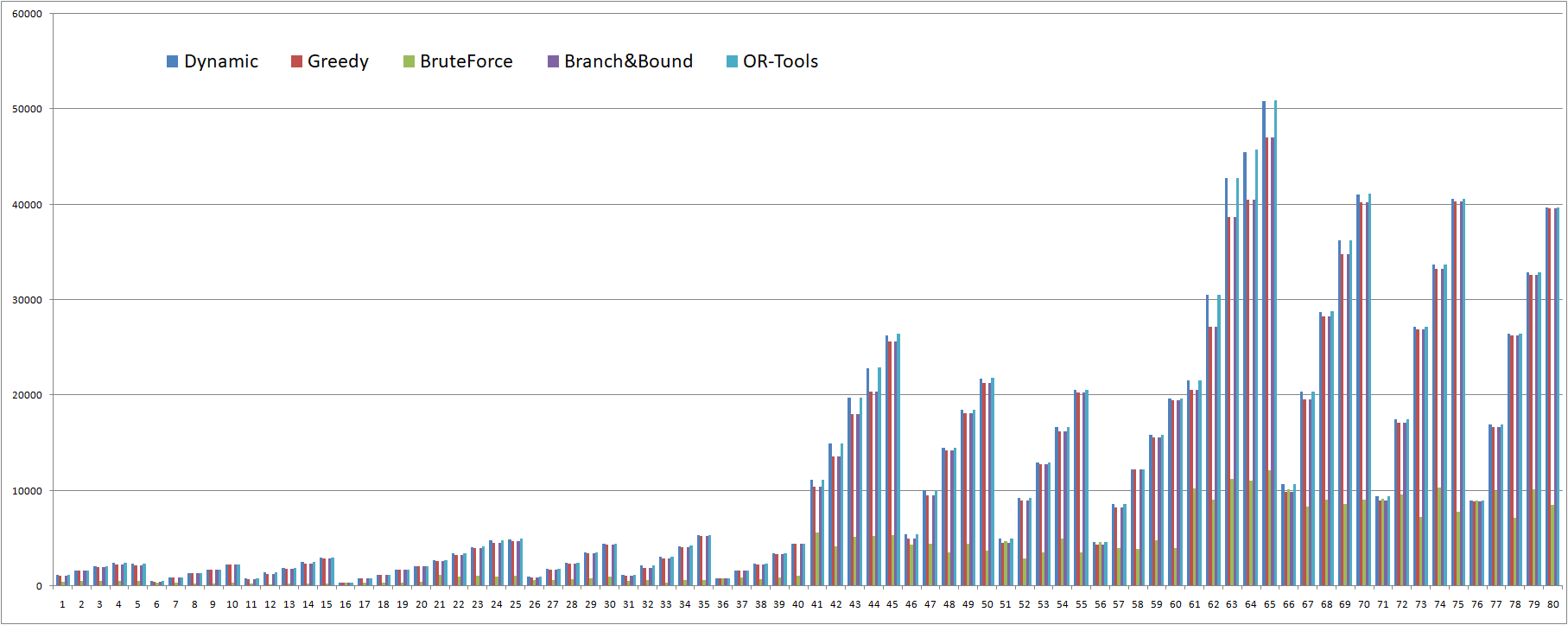
**ΓΡΑΦΗΜΑ 1 . Αποτελέσματα αξιών σακιδίου για πλήθος αντικειμένων ν=10, 80 περιπτώσεις**

Αυξάνοντας τώρα το πλήθος των αντικειμένων σε (ν=50) ακόμα και για μικρές τιμές χωρητικότητας σακιδίου παρατηρούμε ότι οι αλγόριθμοι αρχίζουν να έχουν αποκλίσεις στις τιμές που επιστρέφουν, με κάποιες φορές οριακά καλύτερο τον επιλυτή OR-TOOLS, και κάποιες άλλες τον Dynamic ακολουθούμενο από τους Branch and Bound, και Greedy, ενώ παρατηρούμε ότι ο Brute Force αρχίζει να μην πετυχάινει καθόλου καλές τιμές αξιών γιατί λήγει το χρονικό όριο των 10 δευτερολέπτων που του έχει διατεθεί για εκτέλεση. Αυτό φαίνεται περισσότερο στα παραδείγματα με μεγαλύτερες αξίες σακιδίου (από το 35 περίπου, έως και το 80, Βλ. Γράφημα2).



**ΓΡΑΦΗΜΑ 2 . Αποτελέσματα αξιών σακιδίου για πλήθος αντικειμένων ν=50, 80 περιπτώσεις**

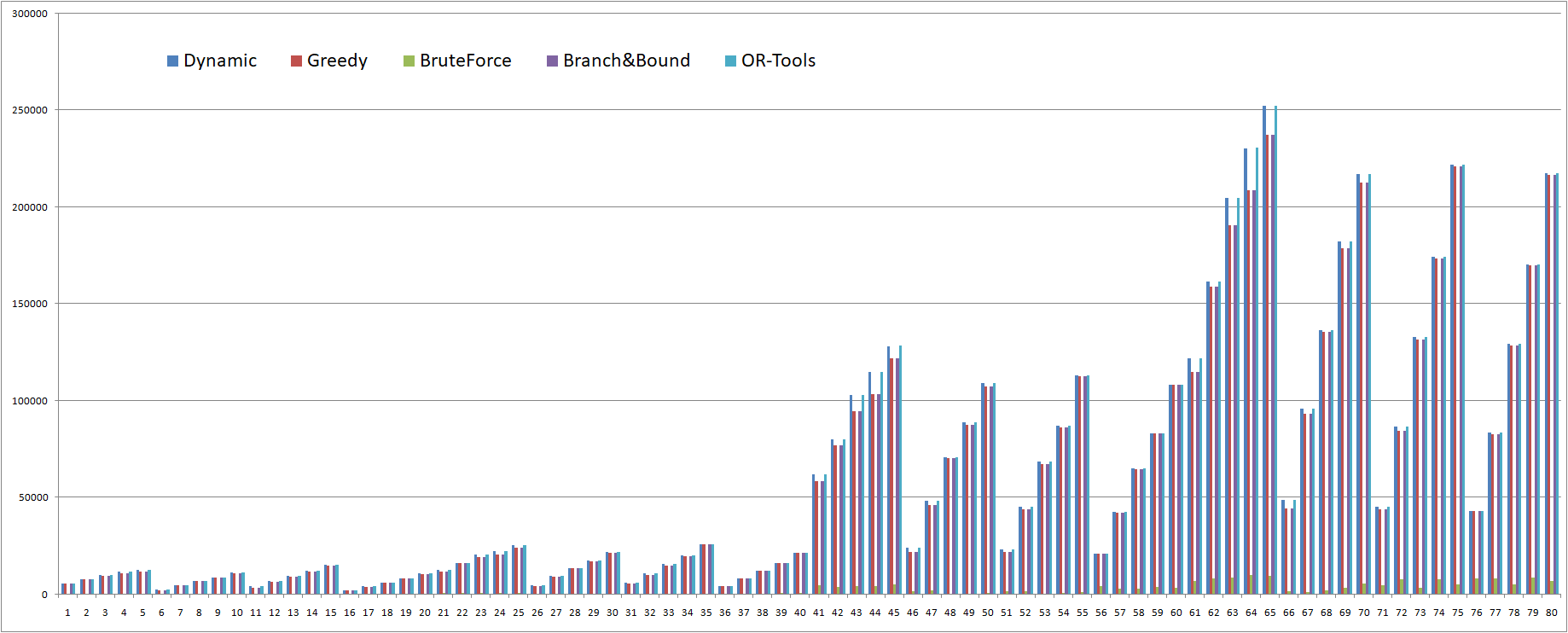
Αυξάνοντας ακόμα περισσότερο το πλήθος των αντικειμένων σε (ν=100) ακόμα και για μικρές τιμές χωρητικότητας σακιδίου παρατηρούμε ότι οι αλγόριθμοι δεν έχουνσοβαρές αποκλίσεις στις τιμές που επιστρέφουν για τις τιμές μικρής συνολικής χωρητικότητας σακιδίου (δηλαδή τα αρχεία από 1-40) με κάποιες φορές οριακά καλύτερο τον επιλυτή OR-TOOLS, και κάποιες άλλες τον Dynamic ακολουθούμενο από τους Branch and Bound, και Greedy. Εδώ παρατηρούμε ότι από την αρχή ο Brute Force δεν μπορεί να δώσει καθόλου αξιόπιστο αποτέλεσμα γιατί δεν προλαβάινει να εκτελεσετεί. Θυμίζουμε ότι ο Brute Force αλγόριθμος συνδυάζει μια τιμή με όλες τις υπόλοιπες με συνέπεια να χρειάζεται στο συγκεκριμένο παράδειγμα 2100 συνδυασμούς για να παράξει αποτέλεσμα, συνεπώς δεν υπάρχει καμμιά περίπτωση να ικανοποιηθεί η συνθήκη του χρόνου των 10 δευτερολέπτων και πρακτικά να μη βρίσκει ποτέ λύση. Στο δεύτερο μισό του γραφήματος βλέπουμε ότι οι Dynamic και OR\_TOOLS αρχίζουν να παράγουν πολύ πιο ικανοποιητικό αποτέλεσμα από τους υπόλοιπους. (Βλ. Γράφημα 3)



**ΓΡΑΦΗΜΑ 3 . Αποτελέσματα αξιών σακιδίου για πλήθος αντικειμένων ν=100, 80 περιπτώσεις**

Όσο μεγαλώνουν ακόμα περισσότερο οι τιμές του ν, παρατηρούμε ότι οι αλγόριθμοι αποδίδουν ταυτόσημα αποτελέσματα με ενναλαγές στην απόδοσή τους μεχρι περίπου το μισό του δπλήθους του δείγαμτος, αλλά πάντα οι Dynamic και OR-TOOLS πετυχαίνουν καλύτερα αποτελέσματα έστω και με μικρή διαφορά.

Καταλαβαίνουμε ότι ο Brute Force πρέπει να αντιμετωπίσει 2500 συνδασμούς με συνέπεια να μη βρίσκει καθόλου λύση. Στο δεύτερο μισό του πλήθους του δείγματος(από το αρχείο 40 και πάνω) παρατηρούμε ότι η διαφορά μεταξύ των Dynamic και OR-TOOLS και τους υπόλοιπους παρουσιάζει σχετική αύξηση (Βλ. Γράφημα 4).



**ΓΡΑΦΗΜΑ 4 . Αποτελέσματα αξιών σακιδίου για πλήθος αντικειμένων, ν=500, 80 περιπτώσεις**

Είναι σημαντικό να αναφερθούμε και στους χρόνους εκτέλεσης των αλγορίθμων.

Μέσα από τα .csv αρχεία παρατηρούμε ότι σε όλες τις περιπτώσεις πιο γρήγορος αλγόριθμος με διαφορά είναι ο Greedy. Επίσης συμπεράνουμε ότι οι Branch and Bound, καθώς και η OR-TOOLS επιστρέφουν σε λίγο χρόνο αποτελέσματα αρκετά ικανοποιητικά με σχετική ακρίβεια. Φυσικά ο πιο αργός αλγόριθμος είναι ο Brute Force.

**ΣΥΜΠΕΡΑΣΜΑΤΑ**: Παρατηρούμε ότι για το συγκεκριμένο πρόβλημα καλύτερη μέθοδος αποδεικνύεται ο αλγόριθμος του εξειδικευμένου επιλυτή της Google, ο OR-TOOLS. Πολύ κοντά του σε απόδοση βρίσκεται ο Dynamic Programming αλγόριθμος και ακολουθούν με σχεδόν ταυτόσημα αποτελέσματα οι Greedy (ο οποίος όμως είναι ο πιο γρήγορος) και Branch and Bound.

Ο αλγόριθμος Brute Force λόγω της ιδιαιτερότητάς του να ελέγχει όλους τους πιθανούς συνδυασμούς αποδίδει εξαιρετικά καλά αποτελέσματα όσο το πλήθος του δείγματος είναι μικρό (ν=10). Αυξανόμενο το δείγμα σε ν=50 αρχίζει και αδυνατεί να εξετάσει όλες τις περιπτώσεις στον χρόνο που μας δόθηκε ως όριο δηλαδή τα 10 sec.

**ΑΠΟΘΕΤΗΡΙΟ**:

Οι κώδικες , τα αποτελέσματα και όλα τα απαιτούμενα αρχεία για την εργασία βρίσκονται αναρτημένα

στο αποθετήριο που είναι προσβάσιμο από το εξής link: <https://github.com/vasilisloupas/Knapsack>

**ΙΣΤΟΣΕΛΙΔΑ**:

Αποτελέσματα και σύνοψη της εργασίας παρουσιάζεται στην ιστοσελίδα του παρακάτω

συνδέσμου: <https://vasilisloupas.github.io>